# Quiz 3: Consensus

$$(x+y)(y'+z) = (x+y)(y'+z)(x+z)$$

Consensus Variable

(Redundant) Consensus Term

consensus variable

$$(x'+y)(y'+z)(x+z') = (x'+y)(y'+z)(x+z')(x'+z)(y+z')(x+y')$$

- 3部分两两组合都能形成一个 consensus term

$$= (x'+y)(y'+z)(x+z')(x'+z)(y+z')(x+y')$$

- 之后原始的三部分和 consensus term 形成的三部分组成6部分

$$= (x'+z)(y+z')(x+y')$$

- 发现若以 consensus term 为基准，之前原始的部分可以看作是 consensus term 和 consensus term 可以被省略掉。

- 也就是说，6项之乘积，可以被前三项之积或后三项之积所替代。

# What are minterms???

· The connection between minterm index and where the minterm is equal to "1" is to look at the index as a row number

minterm的 index 和 row index 一致时为1

decimal ⇔ 对应的 binary number.

| Row Index | $x$ | $y$ | $m_0$ | $m_1$ | $m_2$ | $m_3$ |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 | 1 | 0 | 0 |
| 2 | 1 | 0 | 0 | 0 | 1 | 0 |
| 3 | 1 | 1 | 0 | 0 | 0 | 1 |

Symbolically:

| | | | |
|---|---|---|---|
| $x'y'$ | $x'y$ | $xy'$ | $xy$ |

The minterm is a simple elementary function that has the property that is "true" or equal to "1" for a singular combination of the variable. (just true for one row)

The reason why we don't care is
- that combination will not happen
- the output don't matter

incomplete specification.

# Don't Cares

| Decimal | x y z | f |
|---|---|---|
| 0 | 000 | 1 |
| 1 | 001 | 0 |
| 2 | 010 | d |
| 3 | 011 | 1 |
| 4 | 100 | 0 |
| 5 | 101 | d |
| 6 | 110 | 1 |
| 7 | 111 | 1 |

It could be "0" or "1"
It helps when we try to minimize, because we can make d as "0"

$$f(x,y,z) = \sum_{x,y,z}(0,3,6,7) + d(2,5)$$

$$f(x,y,z) = \prod_{x,y,z}(1,4)d(2,5)$$

把这些 index 组成的 minterm 带入 switch function, 得到的都是"1".

On-Set $= \{0, 3, 6, 7\}$ ⟶ minterm be "1"
Off-Set $= \{1, 4\}$ ⟶ minterm be "0"
Don't-Care-Set $= \{2, 5\}$ ⟶ minterm could be either

# Code Word Representation of Product Terms

Variables: $u\ v\ w\ x\ y\ z$

| Code Word | Product Term | #minterms "covered" |
|---|---|---|
| 001101 | $u'v'wxy'z$ | $1\ \{m_{13}\}$ |
| 0-1-01 | $u'wy'z$ | $2^2 = 4\ \{m_9, m_{13}, m_{25}, m_{29}\}$ |

001001 001101 011001 011101

dash means the corresponding variable is missing

It's not an minterm anymore.

since the missing variable could be 0. 1.
所以有四种可能的组合.

因此也对应可能的四个 minterm.

$$\#\text{minterms covered by code word} = 2^{\#\text{missing literals}}$$

# Boole's (Shannon's) Expansion Theorem

- Decomposition of a switching function of $n$ variables into functions of $n-1$ variables:

$$f(x_1, x_2, \cdots, x_n) = x_1' f_{x_1'} + x_1 f_{x_1}$$

$$f(x_1, x_2, \cdots, x_n) = [x_1' + f_{x_1}] \cdot [x_1 + f_{x_1'}]$$

- Repeated application yields canonical forms (ex: $n = 2$)

$$f(x,y) = x'f(0,y) + xf(1,y)$$

$$f(x,y) = x'[y'f(0,0) + yf(0,1)] + x[y'f(1,0) + yf(1,1)]$$

$$f(x,y) = f(0,0) \cdot x'y' + f(0,1) \cdot x'y + f(1,0) \cdot xy' + f(1,1) \cdot xy$$

$$f(x,y) = f_{x'y'} \cdot x'y' + f_{x'y} \cdot x'y + f_{xy'} \cdot xy' + f_{xy} \cdot xy$$

通用表达式
generic expression of any function of two variables

$$f(x,y) = a_0 \cdot m_0 + a_1 \cdot m_1 + a_2 \cdot m_2 + a_3 \cdot m_3$$

the value of function at (0,0)

$a_0 - a_3$ 可为0或1，所以 $f(x,y)$ 可以有16种展开方式.

# Truth Table for *Arbitrary* 2-Variable Function $f(x, y)$

$$f(x,y) = a_0 m_0 + a_1 m_1 + a_2 m_2 + a_3 m_3$$
$$= a_0 x'y' + a_1 x'y + a_2 xy' + a_3 xy$$

| Row Index | $x$ | $y$ | $f$ |
|---|---|---|---|
| 0 | 0 | 0 | $a_0$ |
| 1 | 0 | 1 | $a_1$ |
| 2 | 1 | 0 | $a_2$ |
| 3 | 1 | 1 | $a_3$ |

Think of each $a_i$ as a *minterm selector*

$$a_i = \begin{cases} 1, m_i \text{ is a minterm of } f \\ 0, m_i \text{ is not a minterm of } f \end{cases}$$

# Algebraic v. Set View of Functions

$$f(x,y) = a_0 m_0 + a_1 m_1 + a_2 m_2 + a_3 m_3$$
$$= a_0 x'y' + a_1 x'y + a_2 xy' + a_3 xy$$

$a_1$ 0, $a_3$ 0

so the function become only "minterm 0" and "minterm 2"

$$a_0 = a_2 = 1 \Rightarrow \quad f(x,y) = m_0 + m_2$$
$$= x'y' + xy'$$
$$= (x' + x)y'$$
$$= y'$$

$$U = \{m_0, m_1, m_2, m_3\}$$
$$f(x,y) \subseteq U$$

f(x,y) 的表达式由 U 的 subset 组成.

U 的 subset 有16种可能

$$f(x,y) = \{m_0, m_2\}$$

1个 Variable 有几种 Switching function 的可能.

- 公式 $2^{2^n} = 2^2 = 4$.
- $f(x) = 1$ ; $f(x) = 0$ ; $f(x) = x$ ; $f(x) = x'$
- 通过 f(x) generic solution 的方法推理.

$f(x) = x'f(0) + xf(1)$ also $= (x' + f(1))(x + f(0))$

$= x'f_{x'} + xf_x$

$= a_0 x' + a_1 x$

for the combination of $a_0$ and $a_1$, there are total four combination possibility.

# Functional Properties

- The canonical SOP (POS) form is unique, subject to commutativity
- Two functions are logically equivalent iff their canonical SOP (POS) form are identical
- The canonical SOP (POS) form contains $2^n$ coefficients each of which can be either 0 or 1. Thus, there are $2^{2^n}$ switching functions of $n$ variables

中有很多可以被省略号：(x+y') 和 (x'+y)

| $n$ | $2^n$ | $2^{2^n}$ | $N_n$ |
|---|---|---|---|
| 1 | 2 | 4 | 3 |
| 2 | 4 | 16 | 6 |
| 3 | 8 | 256 | 22 |
| 4 | 16 | 65,536 | 402 |
| 5 | 32 | 4,294,967,296 | 1,228,158 |

$N_n$ = number of "types" of functions of $n$ variables [Slepian 53][a]

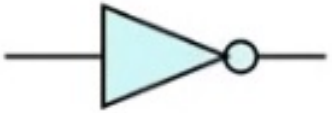
[a] Slepian, D., "On the Number of Symmetry Types of Boolean Functions of $n$ Variables," Can. J. Math., 5(2):185–193, 1953.

# Switching Functions of 2 Variables

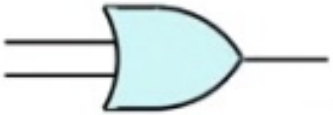
$$f(x,y) = a_0 m_0 + a_1 m_1 + a_2 m_2 + a_3 m_3$$

| $a_3 a_2 a_1 a_0$ | $f(x,y)$ | Name | Symbol | Unique? |
|---|---|---|---|---|
| 0000 | 0 | Inconsistency | | ‡ |
| 0001 | $x'y'$ | NOR | $x \downarrow y$ | |
| 0010 | $x'y$ | Inhibition | | |
| 0011 | $x'$ | NOT | | |
| 0100 | $xy'$ | Inhibition | | |
| 0101 | $y'$ | NOT | | |
| 0110 | $x'y + xy'$ | XOR | $x \oplus y$ | ‡ |
| 0111 | $x' + y'$ | NAND | $x \uparrow y$ | |
| 1000 | $xy$ | AND | $x \cdot y = x \wedge y = xy$ | ‡ |
| 1001 | $xy + x'y'$ | XNOR / EQV | $x \odot y$ | |
| 1010 | $y$ | Transfer | | |
| 1011 | $x' + y$ | Implication | $x \rightarrow y$ | |
| 1100 | $x$ | Transfer | | ‡ |
| 1101 | $x + y'$ | Implication | $y \rightarrow x$ | |
| 1110 | $x + y$ | OR | $x + y = x \vee y$ | ‡ |
| 1111 | 1 | Tautology | | ‡ |

# Logic Gates

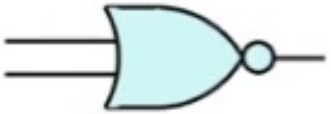
NOT $x$  $x'$

AND $x$, $y$  $xy$

OR $x$, $y$  $x + y$

XOR $x$, $y$  $x \oplus y = x'y + xy'$

NAND $x$, $y$  $(xy)' = x' + y'$

NOR $x$, $y$  $(x + y)' = x'y'$

XNOR $x$, $y$  $x \odot y = x'y' + xy$

# Properties of XOR (modulo-2 Addition)

- Commutativity: $x \oplus y = y \oplus x$
- Associativity: $(x \oplus y) \oplus z = x \oplus (y \oplus z)$
- Distributivity: $x(y \oplus z) = xy \oplus xz$
- Relationship to XNOR: $(x \oplus y)' = x \odot y$
  - XNOR is "equal"
  - XOR is "not equal"
- Conditional Complementation:

$$s \oplus x = \begin{cases} x & \text{if } s = 0 \\ x' & \text{if } s = 1 \end{cases}$$

$S \oplus x = S\cancel{x} + S\cancel{x}'$ ✓

奇偶性

- Parity: Value of $f = x_1 \oplus x_2 \oplus \ldots \oplus x_n$
  - Remains unchanged if an even number of variables are complemented
  - Is complemented if an odd number of variables are complemented
- Any identity $f(X) = g(X)$ can be re-expressed as $f(X) \oplus g(X) = 0$

So XOR are used in parity computations

checking the number of "1" in a sequence of bits

- the number of "1" is odd, then parity is odd
- the number of "1" is even, then parity is even

# Functional Completeness

- A set of operations is functionally-complete (or universal) iff every switching function can be expressed entirely by means of operations from this set
- The following are functionally-complete operation sets
  - $\{+,\cdot,'\}$ (by definition)
  - $\{+,'\}$ (by De Morgan's theorem)
  - $\{\cdot,'\}$ (by De Morgan's theorem)
  - {NAND}: $x \uparrow y = x' + y'$
    Complement: $x \uparrow x = x' + x' = x'$
    AND: $(x \uparrow y) \uparrow (x \uparrow y) = (x \uparrow y)' = (x' + y')' = xy$
  - {NOR}

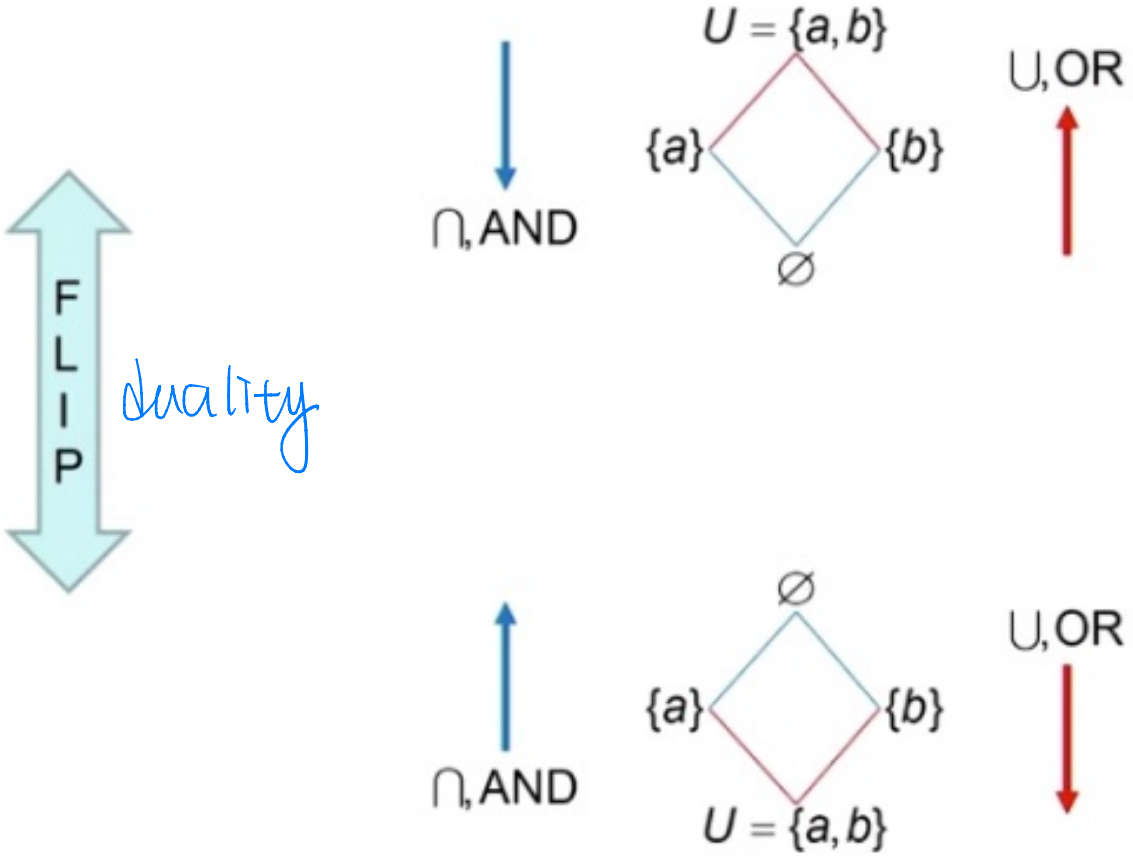
# Isomorphic Systems

means you can establish a 1 to 1 correspondence (对应) between the elements and the operations from a system to the other. Only different is you name them differently.

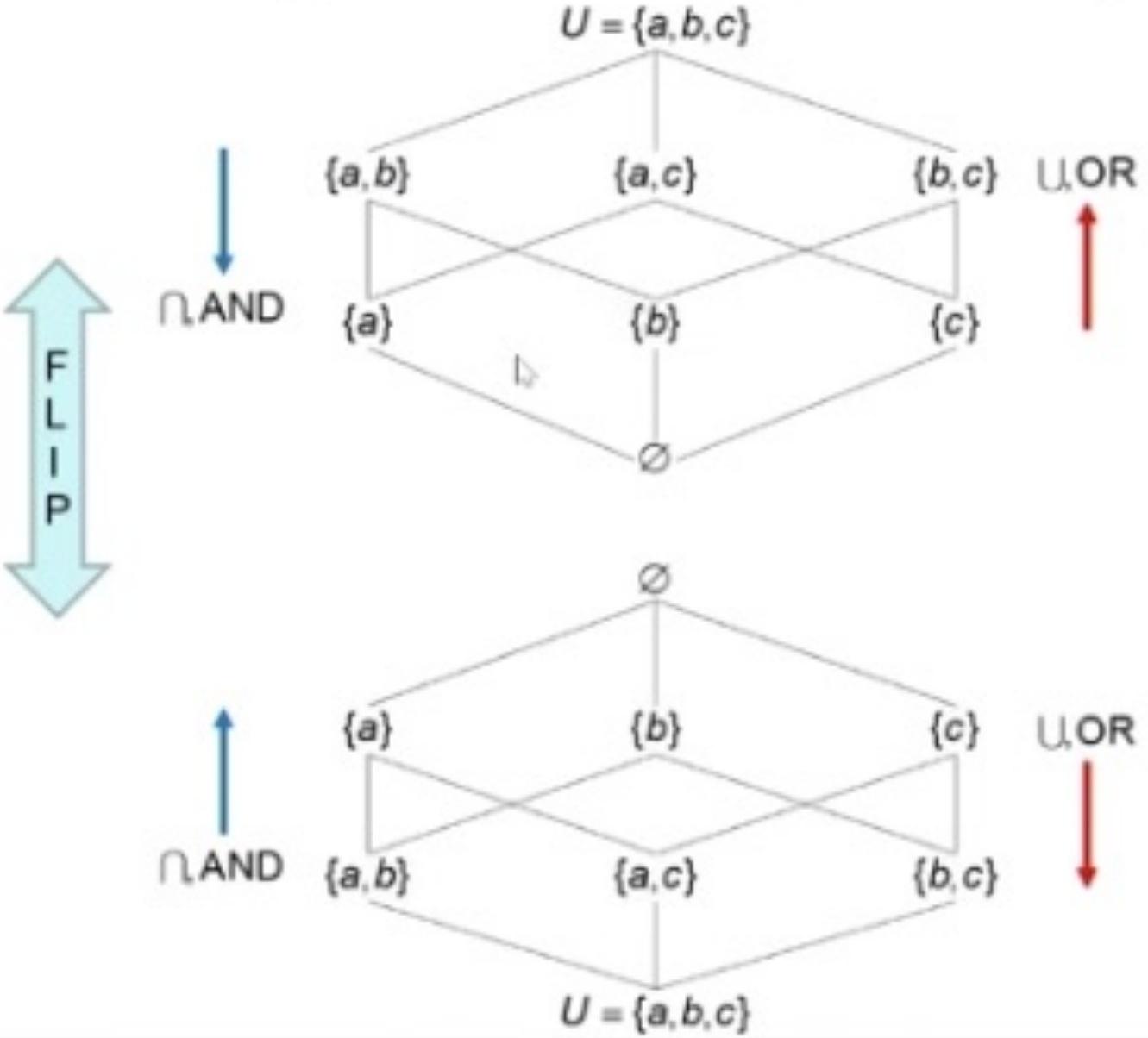
- Two algebraic systems are isomorphic if there is a one-to-one correspondence between elements and operations from one system to the other
- Examples of algebraic systems which are isomorphic to Switching algebra:
  - Series-parallel switching circuits
  - Propositional calculus
  - Algebra of sets

| | Switching Algebra | Algebra of Sets |
|---|---|---|
| Elements | 0 | $\emptyset$ empty set |
| | 1 | $U$ universal set |
| Unary Op | $x'$ | $U - x$ |
| Binary Ops | $x \cdot y$ | $x \cap y$ intersection |
| | $x + y$ | $x \cup y$ union |

# ``Big'' Boolean Algebras: Duality



# ``Big'' Boolean Algebras: Duality



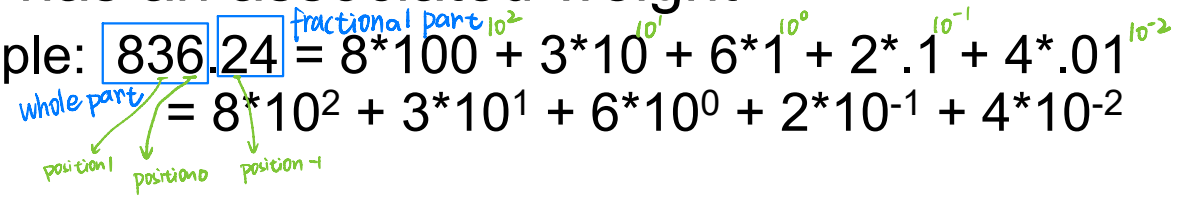
# UM EECS 270 F22
# Introduction to Logic Design

## 6. Binary Representation of Positive Numbers

bits mean nothing unless we give them meaning

# Representation of Positive Numbers

- So far we've been looking at binary signals, yet most concepts in the real world are represented as base-10 (decimal) numbers. How can we represent these concepts on a computer? First let's fully understand our "normal" number system…
- We use a **positional number system**; one where a number is represented by a string of digits and each digit position has an associated weight
  - Example: 836.24 = $8*100 + 3*10 + 6*1 + 2*.1 + 4*.01$
    $= 8*10^2 + 3*10^1 + 6*10^0 + 2*10^{-1} + 4*10^{-2}$




*10 is called the **radix** or **base***

*The "decimal point" is more generally referred to as the **radix point***

What is an example of a non-positional number system?

- Example: In radix 3 with digits {0, 1, 2}:

$$1021_3 = 1*3^3 + 0*3^2 + 2*3^1 + 1*3^0 = 34_{10}$$

- In general, for any radix *r*, the value of a number written

$$d_{p-1}d_{p-2}...d_0.d_{-1}d_{-2}...d_{-n}$$

can be expressed as the weighted sum:

$$\sum_{i=-n}^{p-1} d_i \cdot r^i$$

where $d_i$ is a member of a set of digits

- In general, the lower the radix, the more digits are needed to represent a number
- Useful radices:
  - 10/decimal (10 fingers!)
  - 2/binary (0/1 – high/low voltage)
  - 8/octal and 16/hexadecimal (compact representation of binary)

用于表达更长的2进制数字

- **Binary** (base-2): digits {0, 1}

$$1011.101_2 = 1*2^3 + 0*2^2 + 1*2^1 + 1*2^0 + 1*2^{-1} + 0*2^{-2} * 1*2^{-3}$$
$$= 8 + 2 + 1 + .5 + .125 = 11.625_{10}$$

- Large numbers require lots of digits!

$$27825_{10} = 110110010110001_2$$

- Often convenient to use octal or hexadecimal as a shorthand for binary

- **Octal** (base-8): digits {0, 1, 2, 3, 4, 5, 6, 7}

$$321_8 = 3*8^2 + 2*8^1 + 1*8^0 = 209_{10}$$

- Because 8 is a power of 2 ($2^3$), conversion from binary to octal and vice-versa is easy! Each octal digit represents exactly 3 binary digits.

Octal to binary: $6\ \ 2\ \ 1_8 =$

$110\ 010\ 001_2$

*Make sure to pad with 0s when necessary!*

Binary to octal: $001\ 011.110\ 1\mathbf{00}_2 =$

$1\ \ 3\ .\ 6\ \ 4_8$

- Hexadecimal (base-16):

  digits {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F}

  (can't borrow digits for 10-15 from base-10, so use A-F)

  $$8B7_{16} = 8*16^2 + 11*16^1 + 7*16^0 = 2231_{10}$$

- Because 16 is a power of 2 ($2^4$), each hexadecimal digit represents 4 binary digits

  Hexadecimal to binary: A $8_{16}$ =

  $1010\ 1000_2$

  Binary to hexadecimal: $0110\ 1110_2$ =

  6 $E_{16}$

- Often hex numbers are denoted with the prefix "0x" instead of a subscript 16 (0x3B8 ≡ $3B8_{16}$)
- How to convert from hex to octal and octal to hex??
  - Go through binary!
- Why hex? $27825_{10} = 110110010110001_2 = 6CB1_{16}$

- So far we know the following base conversions:
  - binary, octal, hex → decimal
  - binary ↔ octal ↔ hex
- What about decimal → binary, octal, or hex?

*Use repeated division by radix:*

$178_{10} = ?_2$

| | | |
|---|---|---|
| 2 \| 178 | | |
| 2 \| 89 | R 0 | LSB |
| 2 \| 44 | R 1 | |
| 2 \| 22 | R 0 | |
| 2 \| 11 | R 0 | |
| 2 \| 5 | R 1 | |
| 2 \| 2 | R 1 | |
| 2 \| 1 | R 0 | |
| 0 | R 1 | MSB |

$178_{10} = 10110010_2$

$178_{10} = ?_8$

| | | |
|---|---|---|
| 8 \| 178 | | |
| 8 \| 22 | R 2 | LSB |
| 8 \| 2 | R 6 | |
| 0 | R 2 | MSB |

$178_{10} = 262_8$

$178_{10} = ?_{16}$

| | | |
|---|---|---|
| 16 \| 178 | | |
| 16 \| 11 | R 2 | LSB |
| 0 | R 11 (B) | MSB |

$178_{10} = B2_{16}$

# Why does repeated division work?

$$N_r = (d_3*r^3 + d_2*r^2 + d_1*r^1 + d_0r^0)_{10}$$

(remember that $d_i < r$)

| | | |
|---|---|---|
| $N_r$ | $d_3*r^3 + d_2*r^2 + d_1*r^1 + d_0r^0$ | |
| Divide by *r*: | $d_3*r^2 + d_2*r^1 + d_1r^0$ | remainder $d_0$ |
| Divide again by *r*: | $d_3*r^1 + d_2r^0$ | remainder $d_1$ |
| Divide again by *r*: | $d_3r^0$ | remainder $d_2$ |
| Divide again by *r*: | 0 | remainder $d_3$ |

# Fraction conversion: use repeated *multiplication* by radix

在10进制中，不是所有小数都能被其它进制所表示的。

*Watch for repeating digits…*

$.59375_{10} = ?_2$

| | | |
|---|---|---|
| | .59375 | * 2 |
| MSB 1 | .1875 | * 2 |
| 0 | .375 | * 2 |
| 0 | .75 | * 2 |
| 1 | .5 | * 2 |
| LSB 1 | .0 | |

用 fraction part 和 dtax number 相乘.

$.59375_{10} = .10011_2$

$.7_{10} = ?_2$

| | |
|---|---|
| | .7 * 2 |
| 1 | .4 * 2 |
| 0 | .8 * 2 |
| 1 | .6 * 2 |
| 1 | .2 * 2 |
| 0 | .4 * 2 |

这一部分会循环

$.7_{10} = .\overline{10110}_2$

*Sometimes conversion won't be exact…*

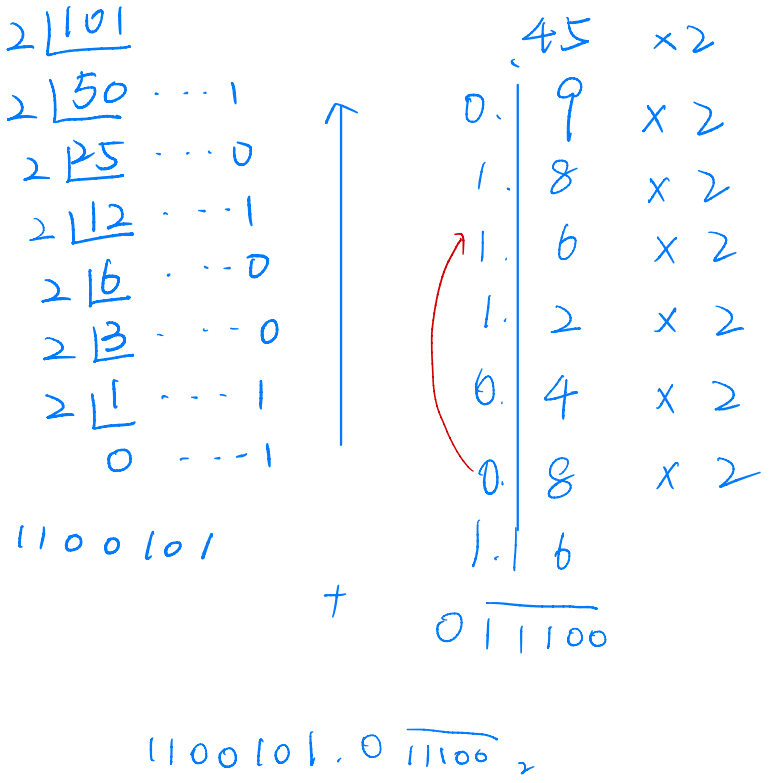
$.73_{10} = ?_2$

| | | |
|---|---|---|
| | .73 | * 2 |
| MSB 1 | .46 | * 2 |
| 0 | .92 | * 2 |
| 1 | .84 | * 2 |
| 1 | .68 | * 2 |
| 1 | .36 | * 2 |
| 0 | .72 | * 2 |
| LSB 1 | .44 | * 2 |

$.73_{10} \approx .1011101_2$

# In Class Exercise

$$101.45_{10} = ?_2, ?_{16}$$




$101.45_{10} = 1100101.01\overline{1100}$

2+4 1+4

这种情况，直接行列恢复写出循环

$= 6\ 5.0111\overline{0011}$

4+2+1 3

7

在保证循环数完整出现一次的前提下，行和写满4的倍数，然后补齐前循环

$= 6\ 5.\overline{73}_{16}$

# In Class Exercise

$101.45_{10} = ?_2, ?_{16}$

$101_{10} = ?_2$

2 | 101
2 | 50 R 1 LSB
2 | 25 R 0
2 | 12 R 1
2 | 6 R 0
2 | 3 R 0
2 | 1 R 1
0 R 1 MSB

$101_{10} = 1100101_2$

$.45_{10} = ?_2$

.45 * 2
0 . 9 * 2
1 . 8 * 2
1 . 6 * 2
1 . 2 * 2
0 . 4 * 2
0 . 8 * 2

$.45_{10} = .01\overline{1100}_2$

$101.45_{10} = 1100101.01\overline{1100}_2$

$0110\ 0101.0111\ \overline{0011}_2$
$= 6\quad 5\ .\ 7\quad \overline{3}_{16}$
$= 65.7\overline{3}_{16}$

# Equality Comparator

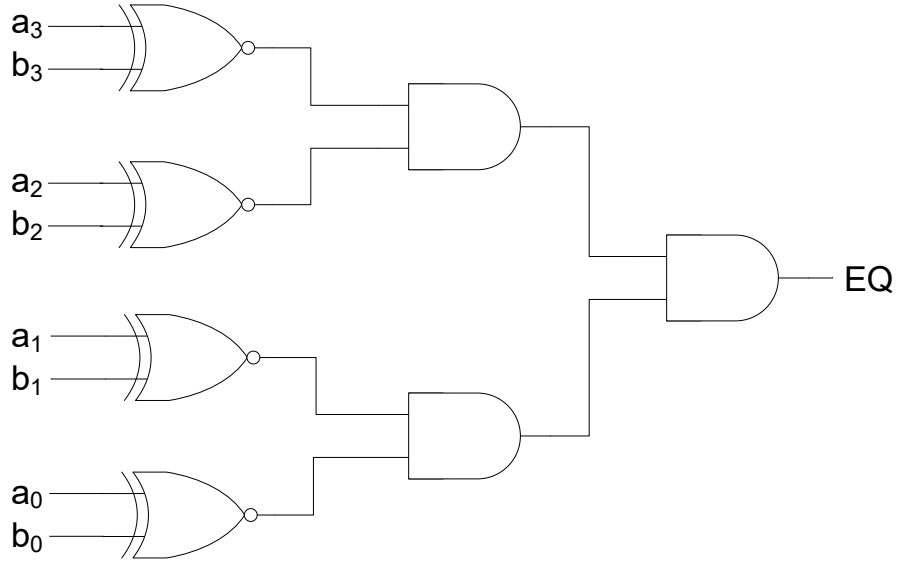
- Want a signal, EQ, that is 1 iff two 4-bit numbers, A and B, are equal. Under what logic conditions are A and B equal?
  - Equal if $a_3 = b_3$ AND $a_2 = b_2$ AND $a_1 = b_1$ AND $a_0 = b_0$

corresponding bits 必须相等 → XNOR告诉我们两个东西是否相等

| A | B | A⊙B |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

- Which gate performs an equality comparison?
  - XNOR!

$$EQ = (a_3 \odot b_3) \cdot (a_2 \odot b_2) \cdot (a_1 \odot b_1) \cdot (a_0 \odot b_0)$$




How many gates are required to implement an *n*-bit "parallel" comparator?

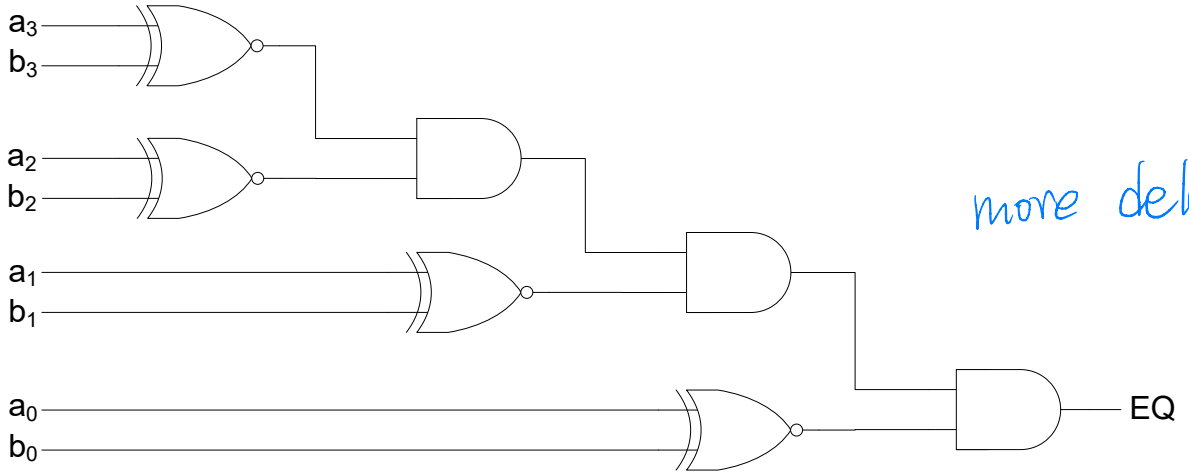
Assume $n = 2^k$

#gates = $2^k + 2^{k-1} + 2^{k-2} + \ldots 2^0$

$= 2^{k+1} - 1$

$= 2n - 1$

# A “Series” Comparator Implementation




How many gates are required to implement an *n*-bit “series” comparator?

- Comparing first two bits: 3 gates

- Comparing each additional bit: 2 gates

- After comparing the first 2 bits, there are n-2 bits left to compare for a total of 2*(n-2) gates

-Total number of gates required:  2(n-2) + 3 = 2n – 1

- “Series” and “parallel” implementations require the same number of gates!  Is there a reason to prefer one implementation over the other?

# Magnitude Comparator

- Want to design a signal, AgrB, that is 1 iff A > B (A and B are 4-bit numbers). Under what logic conditions is A > B?
  - A > B if $a_3 = 1$ and $b_3 = 0$
  - A > B if $a_2 = 1$ and $b_2 = 0$ *and* $a_3 = b_3$
  - A > B if $a_1 = 1$ and $b_1 = 0$ and $a_2 = b_2$ and $a_3 = b_3$
  - A > B if $a_0 = 1$ and $b_0 = 0$ and $a_1 = b_1$ and $a_2 = b_2$ and $a_3 = b_3$

$$AgrB = a_3 \cdot \overline{b_3} +$$
$$(a_3 \odot b_3) \cdot a_2 \cdot \overline{b_2} +$$
$$(a_3 \odot b_3) \cdot (a_2 \odot b_2) \cdot a_1 \cdot \overline{b_1} +$$
$$(a_3 \odot b_3) \cdot (a_2 \odot b_2) \cdot (a_1 \odot b_1) \cdot a_0 \cdot \overline{b_0}$$